A Distributed and Parallel Asynchronous Unite and Conquer Method to Solve Large Scale Non-Hermitian Linear Systems with Multiple Right-hand Sides

Xinzhe Wu, Serge Petiton

A Distributed and Parallel Asynchronous Unite and Conquer Method to Solve Large Scale Non-Hermitian Linear Systems with Multiple Right-hand Sides

Xinzhe Wu[a,b,*], Serge G. Petiton[a,b]

[a]*Maison de la Simulation/CNRS, Gif-sur-Yvette, 91191, France*
[b]*Université de Lille, Villeneuve d'Ascq, 59650, France*

Abstract

Many problems in the field of science and engineering often require to solve simultaneously large-scale non-Hermitian sparse linear systems with multiple right-hand sides (RHSs). Efficiently solving such problems on extreme-scale platforms requires the minimization of global communications, reduction of synchronization points and promotion of asynchronous communications. We develop an extension of the Unite and Conquer GMRES/LS-ERAM (UCGLE) method [1] by combining it with Block GMRES method to solve non-Hermitian linear systems with multiple RHSs. UCGLE is a hybrid method consisting of three computing algorithms with asynchronous communication that allow the use of approximate eigenvalues to accelerate to solve of linear systems and to improve their fault tolerance. In this paper, the variant of UCGLE with novel components and manager engine implementations is introduced. This engine is capable of allocating multiple Block GMRES at the same time, each Block GMRES solving the linear systems with a subset of RHSs and accelerating the convergence using the eigenvalues approximated by other eigensolvers. Dividing the entire linear system with multiple RHSs into subsets and solving them simultaneously with different allocated linear solvers allow localizing calculations, reducing global communication, and improving parallel performance. Meanwhile, the asynchronous preconditioning using eigenvalues is able to speed up the convergence and improve the fault tolerance and reusability. Numerical experiments using different test matrices on supercomputer *ROMEO* indicate that the proposed method achieves a substantial decrease in both computation time and iterative steps with good scaling performance.

Keywords: Linear systems, Krylov subspace methods, unite and conquer, multiple right-hand sides, asynchronous communication
2010 MSC: 00A69, 65F10

1. Introduction

In this paper, we consider solving the system

$$AX = B, \tag{1}$$

where $A \in \mathbb{C}^{n\times n}$ is a large, sparse and non-Hermitian matrix of order n, $X = [x_1, \cdots, x_s] \in \mathbb{C}^{n\times s}$ and $B = [b_1, \cdots, b_s] \in \mathbb{C}^{n\times s}$ are rectangular matrices of dimension $n \times s$ with $s \leq n$. In this paper, the rectangular matrices such as B is also called multi-vector, which can be seen as the combination of s vectors b_i $\forall i \in 1, 2, \cdots, s$. This kind of linear systems with multiple RHSs arise from a variety of applications in different scientific and engineering fields, such as the Lattice Quantum Chromodynamics (QCD) [2, 3, 4], the wave scattering and propagation simulation [5], dynamics of structures [6, 7, 8], etc. The block Krylov methods are good candidates if we want to solve these large linear systems at the same time because the block methods can expand the search space associated with each RHS and may accelerate the convergence. Another feature of block Krylov methods is that they can be implemented using BLAS3, which improves the locality and reusability of data and reduces the memory requirement on modern computer architectures [9]. The block Krylov methods replace the Sparse Matrix-Vector Multiplication (SpMV) in each iterative step of the conventional Krylov methods with the Sparse Generalized Matrix-Matrix (SpGEMM) Multiplication.

However, nowadays, HPC cluster systems continue to scale up not only the number of compute nodes and central processing unit (CPU) cores but also the heterogeneity of components by introducing graphics processing units (GPUs) and many-core processors. This results in the tendency of transition to multi- and many cores within computing nodes, which communicate explicitly through faster interconnection networks. These hierarchical supercomputers can be seen as the intersection of distributed and parallel computing. Indeed, for a large number of cores, the communication of overall reduction operations and global synchronization of applications are the bottleneck. When solving linear systems by block Krylov methods on large-scale distributed memory platforms, the cost of using BLAS3 operations to enlarge search space and reduce the memory requirement is apparent: the communication bound of SpGEMM in each step of Arnoldi projection damages heavily their performance, which cannot be compensated by the advantages of the block methods.

Even using classic Krylov methods, such as GMRES (Gen-

*Corresponding author at: Maison de la Simulation, CNRS USR3441, Building 565, CEA Saclay, 91191 cedex, Gif-sur-Yvette, France. Tel: +33 169085993.
Email addresses:{xinzhe.wu.etu, serge.petiton}@univ-lille.fr.

eralized Minimal Residual method), to solve a large-scale problem on parallel clusters, the cost per iteration of them becomes the most significant concern, typically caused by communication and synchronization overheads. Consequently, large scalar products, overall synchronization, and other operations involving communication among all cores have to be avoided. The numerical applications should be optimized for more local communication, less global communication and synchronization. That is the reason for the recent tendance to study the communication avoiding techniques for linear algebra operations [10, 11, 12] and different pipelined strategies for Krylov methods [13, 14, 15]. For benefiting the full computational power of such hierarchical systems, it is central to explore novel parallel methods and models for the solving of linear systems. These methods should not only accelerate the convergence but also have the abilities to adapt to multi-grain, multi-level memory, to improve the fault tolerance, reduce synchronization and promote asynchronization.

We propose to combine the Block GMRES (BGMRES) [16] with UCGLE [1] to solve Equation (1) in parallel on modern computer architectures. UCGLE is a hybrid method which composes three computing components with asynchronous communication: ERAM (Explicitly Restarted Arnoldi Method), GMRES and LS (Least Squares) components. GMRES Component is used to solve the systems, and LS Component performs as a preconditioner which uses the eigenvalues approximated by ERAM Component to speed up the convergence. LS Component is implemented based on the Least squares polynomial method proposed by Saad [17] in 1987. Compared with the traditional hybrid methods, the key features of UCGLE are its distributed and parallel asynchronous communications and the implementation of a manager engine among three components, which target at the extreme-scale supercomputing platforms. The asynchronous communication overlaps the overhead of overall synchronization and improves the fault tolerance and reusability. In this paper, firstly, we develop a block version of Least Squares Polynomial (B-LSP) method based on [17], then replace the three computing components of UCGLE respectively by the BGMRES, Shifted Krylov-Schur (s-KS), and B-LSP. Additionally, in order to solve linear systems with multiple RHSs and reduce the global communication, we design and implement a new manager engine to replace the former one in UCGLE. This novel engine allows to allocate and deploy multiple BGMRES and/or s-KS Components at the same time and support their asynchronous communications. Each allocated BGMRES is assigned to solve the linear systems with a subset of RHSs. This extension is denoted as multiple-UCGLE or m-UCGLE even though the ERAM Component is replaced by s-KS method.

The paper is organized as follows. In Section 2, we present the related work, including the basic Krylov methods, BGMRES and UCGLE. The theoretical work and algorithms inside m-UCGLE are presented in Section 3, especially the mathematical definitions fo B-LSP extended from the Least Squares polynomial method. In Section 4, we present the implementation of a newly designed manager engine for m-UCGLE. In Section 5, we give the information of experimental hardware/software and test matrices. We evaluate the convergence performance of our m-UCGLE implementation[1] compared with conventional restarted BGMRES in Section 6 and we show that our approach has up to 9.5× speedup against the classic method. The parallel performance of m-UCGLE presented in Section 7 show its better strong scalability against the classic restarted BGMRES, and the time to the solution can be significantly decreased. We give the conclusions and perspectives in Section 8.

2. Related Work

In this section, we give a glance at the basic Krylov Subspace methods, BGMRES, and UCGLE.

2.1. Krylov Subspace Methods

In linear algebra, the m-order Krylov subspace generated by a $n \times n$ matrix A and a vector b of dimension n is the linear subspace spanned by the images of b under the first m powers of A, that is

$$K_m(A, b) = span(b, Ab, A^2b, \cdots, A^{m-1}b),$$

The Krylov subspace methods are often used to solve large-scale linear systems and eigenvalue problems.

GMRES is an efficient Krylov subspace method to solve non-Hermitian linear systems. It approximates the solution x_m of $Ax = b$ starting from an initial guessed solution x_0, with the minimal residual in a subspace $K_n(A, r_0)$, where $r_0 = b - Ax_0$ is the initial residual and K_n is the n^{th} Krylov subspace spanned by A from r_0. This method was developed by Saad and Schultz in 1986 [18]. If GMRES method is restarted after a number, say m_g, of iterations with x_{m_g} as a new initial guess, to avoid enormous memory and computation requirements with the increase of Krylov subspace projection number, it is called the restarted GMRES. A well-known difficulty of restarted GMRES is that it can suffer from stagnation in convergence when the matrix A is not positive definite as the restarted subspace is often closer to the earlier one. The typical methods are to use preconditioning techniques, to deflate the eigenvalues/eigenvectors, to recycle the Krylov subspaces, etc.

Arnoldi algorithm is a widely used Krylov subspace method to approximate the eigenvalues of large sparse matrices under the Krylov subspace $K_n(A, v)$, where v is an initial given vector. The numerical accuracy of the computed eigenpairs by Arnoldi method depends highly on the size of $K_m(A, v)$ and the orthogonality of selected basis. Generally, the larger the subspace is, the better approximation of the eigenpairs is. The problem is that firstly the orthogonality of the computed basis tends to degrade with each basis extension. Also, the larger the subspace size is, the larger memory is required. Hence available memory may also limit the subspace size, and the achievable accuracy of the Arnoldi process. In order to overcome this, it is necessary to restart the Arnoldi algorithm, with explicit [19] or implicit [20] deflation of the unwanted eigenvalues. Krylov-Schur method proposed by Stewart [21] is another variant of Arnoldi algorithm with an effective and robust restarting scheme and numerical stability.

[1] https://github.com/brunowu/m-UCGLE

2.2. Block GMRES

Algorithm 1 Non-Restarted Block GMRES Algorithm

function BGMRES($input$:$A, m, B, X_0 \in \mathbb{C}^{N\times s}$ of full rank, $output$: X)
  $R = B - AX_0$
  $V_0R_0 := R$ ▷ QR factorization
  for $j = 1, 2, \cdots, m$ **do**
    $U_j = AV_j$
    for $i = 1, 2, \cdots, j$ **do**
      $H_{i,j} = V_i^T U_j$
      $U_j = U_j - V_i H_{i,j}$
    end for
    $U_j = V_{j+1} H_{j+1,j}$
  end for
  $W_m = [V_1, V_2, \cdots, V_m]$, $H_m = \{H_{i,j}\}_{1\le i\le j+1; 1\le j\le m}$
  Find Y_m, s.t. $\|\beta - H_m Y_m\|_2$ is minimized
  $X = X_0 + W_m Y_m$
end function

BGMRES is a variant of GMRES, which aims to solve sparse non-Hermitian linear systems with multiple RHSs at the same time. As shown in Algorithm 1, the procedure of BGMRES is similar to classic GMRES except replacing SpMV in the Arnoldi projection operation with SpGEMM. Block implementation of GMRES also introduces extra scalar works comparing with running multiple times of GMRES in sequence. Regarding numerical performance, the main attraction of BGMRES is that it can enlarge the search space and potentially speed up the convergence. When solving Equation (1) by BGMRES with Krylov subspace size fixed as m, the search space would be:

$$K_m = span(b_1, Ab_1, \cdots, A^{m-1}b_1, \cdots, b_s, Ab_s, \cdots, A^{m-1}b_s)$$

As discussed in Section 1, the first limitation of BGMRES is caused by the instability and bad parallel performance of its SpGEMM on large-scale distributed memory platforms with the increase of RHS number s. Secondly, for special cases, the convergence by BGMRES cannot always be guaranteed even by enlarging the search spaces. Thus the techniques to deflate of eigenpairs should be applied to accelerate the convergence. That is the motivation for us to propose m-UCGLE which can reduce the global communication and accelerate the convergence by the deflation of eigenvalues.

2.3. Unite and Conquer GMRES/LS-ERAM Method

UCGLE is a hybrid method implemented with asynchronous communication. The design of UCGLE is inspired by the Unite and Conquer (UC) approach.

2.3.1. Unite and Conquer Approach

UC approach is proposed by Emad [22] to explore the novel methods for modern computer architectures. It is a model for the design of numerical methods by combining different computing components using asynchronous communication, and all these components working for the same objective. Unite implies the combination of components, and conquer represents they work together to solve one problem. In the UC methods, different parallel computing components work independently, which can be deployed on various platforms such as P2P, cloud and the supercomputer systems, or different computing units on the same platform.

2.3.2. UCGLE Implementation

UCGLE is a kind of UC approach. As presented in Section 1, it composes three computing components: ERAM, GMRES, and LS. GMRES Component is used to solve the systems, LS and ERAM Components work as the preconditioning part. The asynchronous communication of this hybrid method among three components reduces the number of overall synchronization points and minimizes global communication. The workflow of UCGLE with three computing components is given by Figure 1: ERAM Component computes the desired number of dominant eigenvalues, and then sends them to LS Component; LS Component uses these received eigenvalues to generate a new residual vector, and sends it to GMRES Component; GMRES Component uses this residual as a new restarted initial vector for solving non-Hermitian linear systems. Better convergence acceleration and parallel performance of UCGLE for several test matrices compared with preconditioned GMRES are given in [1].

The convergence acceleration of UCGLE is similar to a deflated preconditioner. The difference is that the asynchronous communication among the different computing components of the latter reduces the global communication and synchronization, improves the fault tolerance and the reusability of this method. The three computing components work independently from each other, when errors occur inside of ERAM, GMRES or LS Component, UCGLE can continue to work as a standard restarted GMRES method. The materials for accelerating the convergence are eigenvalues. These values can be improved on the fly, introducing a continuous improvement of convergence while solving the linear systems. With the help of asynchronous communications, approximated eigenvalues by ERAM Component can be stored into a local file and reused for the other solving procedures with the same matrix. UCGLE is a distributed and parallel method which can profit both shared memory and distributed memory of computational architectures with multiple levels of parallelism.

UCGLE is used to solve non-Hermitian with single RHS, thus its GMRES Component should be replaced by a BGMRES algorithm, and a deflation technique by Least Squares polynomial should be extended to support the solving of linear systems with multiple RHSs. Moreover, as shown in Figure 1, the manager engine in [1] is static. In detail, four communicators of MPI_COMM_WORLD are created for the computing components through MPI_Split using the different colors and keys. The asynchronous communication among them is supported by the MPI non-blocking sending and receiving operations of inter-communicators. This former implementation of manager engine cannot fulfill the requirements of m-UCGLE. Indeed, in order to reduce the global communications and localize the

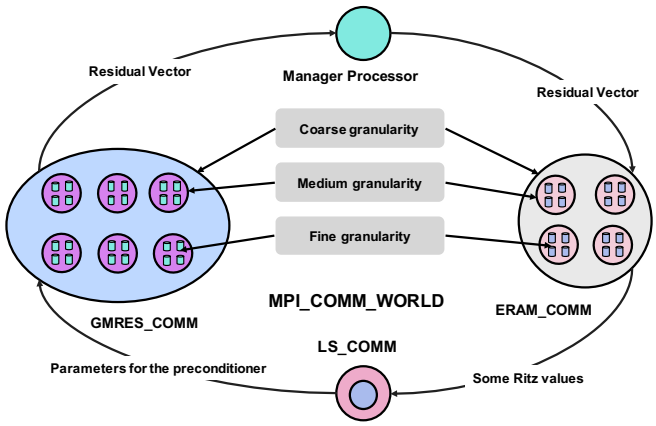

Figure 1: Workflow of UCGLE method [1].

computation, *m*-UCGLE should be able to allocate dynamically different numbers of asynchronous computing components, and each linear solver solves a subset of linear systems.

3. *m*-UCGLE for Multiple Right-hand Sides

The *m*-UCGLE proposed in this paper is an extension of UCGLE and is designed to solve linear systems with multiple RHSs. The three types of components inside UCGLE are respectively replaced by restarted BGMRES that we presented in Section 2.2, *s*-KS, and B-LSP methods. In this section, firstly, we introduce the *s*-KS used in *m*-UCGLE, then the mathematical definitions of B-LSP to support for linear systems with multiple RHSs is given. Finally, we present the numerical analysis and parameter definition of *m*-UCGLE.

3.1. Shifted Krylov-Schur Algorithm

UCGLE uses the dominant eigenvalues to accelerate the convergence of GMRES, and theoretically, the more eigenvalues are applied, the acceleration of Least Squares Polynomial will be more significant [1]. In order to approximate more eigenvalues by ERAM Component, the easiest way is to enlarge the size of related Krylov Subspace. In *m*-UCGLE, we replace ERAM Component by *s*-KS method which is another variant of the Arnoldi algorithm with an effective and robust restarting scheme and numerical stability [21]. The Krylov subspace of *s*-KS cannot be too large. Otherwise BGMRES Component is not able to receive the eigenvalues in time to perform the B-LSP acceleration. With the novel developed manager engine of *m*-UCGLE in this paper, several different *s*-KS components can be allocated at the same time to approximate efficiently the different part of dominant eigenvalues of matrix A, by the shift with different values and thickly restarting with smaller Krylov subspace sizes. The algorithm of *s*-KS is given in Algorithm 2.

Algorithm 2 Shifted Krylov-Schur Method

1: **function** *s*-KS(*input*: A, x_1, m, σ, *output*: Λ_k with $k \leq p$)
2: $A \leftarrow A - \sigma I$
3: Build an initial Krylov decompostion of order m
4: Apply orthogonal transformations to get a Krylov-Schur decompostion
5: Reorder the diagonal blocks of the Krylov-Schur decompostion
6: Truncate to a Krylov-Schur decompostion of order p
7: Extend to a Krylov decomposition of order m
8: If not satisfied, go to step 3
9: **end function**

3.2. Least Squares Polynomial for Multiple Right-hand sides

The Least Squares polynomial method is an iterative method proposed by Saad [17] to solve linear systems. It is applied to calculate a new preconditioned residual for restarted GMRES in UCGLE. In this section, we will present the B-LSP method, which is a block extension of Least Squares polynomial method to solve linear systems with multiple RHSs at the same time. The iterates of B-LSP method can be written as $X_n = X_0 + \mathcal{P}_d(A)R_0$, where $X_0 \in \mathbb{C}^{N\times s}$ is a selected initial guess to the solution, $R_0 \in \mathbb{C}^{N\times s}$ the corresponding residual multi-vector, and $\mathcal{P}_d$ a polynomial of degree $d-1$. We set a polynomial of n degree $\mathcal{R}_n$ such that

$$\mathcal{R}_d(\lambda) = 1 - \lambda\mathcal{R}_d(\lambda)$$

.

The residual of n^{th} steps iteration R_n can be expressed as equation $R_n = \mathcal{R}_d(A)R_0$, with the constraint $\mathcal{R}_d(0) = 1$. We want to find a kind of polynomial which can minimize all $\|\mathcal{R}_d(A)R_0^{(p)}\|_2$, with $p \in 0, 1, \cdots, s-1$, $R_0^{(p)}$ the p^{th} vector in the multi-vector R_0 and $\|.\|_2$ the Euclidean norm.

Suppose A is a diagonalizable matrix with its spectrum denoted as $\sigma(A) = \lambda_1, \cdots, \lambda_n$, and the associated eigenvectors $u_1, \cdots, u_n$. Expanding the each component of R_n in the basis of these eigenvectors as as $R_n^{(p)} = \sum_{i=1}^{n} \mathcal{R}_d(\lambda_i)\rho_i u_i$, which allows to get the upper limit of $\|R_n^{(p)}\|_2$ with $p \in 0, 1, \cdots, s-1$ as:

$$\|R_0^{(p)}\|_2 \max_{\lambda\in\sigma(A)} |\mathcal{R}_d(\lambda)| \tag{2}$$

In order to minimize the norm of $R_n^{(p)}$, it is possible to find a polynomial $\mathcal{P}_d$ which can minimize the Equation (2) $\forall p \in 0, 1, \cdots, s-1$.

As presented in [1], $\mathcal{P}_d$ can be expanded with a basis of Chebyshev polynomial $t_j(\lambda) = \frac{T_j\frac{\lambda-c}{b}}{T_j\frac{c}{b}}$, where t_i is constructed by an ellipse englobing the convex hull formulated by the computed eigenvalues, with c the centre of ellipse, and b the focal distance of this ellipse. $\mathcal{P}_d$ is under form that $\mathcal{P}_d = \sum_{i=0}^{d-1} \eta_i t_i$. The selected Chebyshev polynomials t_i meet still the three terms recurrence relation (3).

$$t_{i+1}(\lambda) = \frac{1}{\beta_{i+1}}[\lambda t_i(\lambda) - \alpha_i t_i(\lambda) - \delta_i t_{i-1}] \tag{3}$$

For the computation of parameters $H = (\eta_0, \eta_1, \cdots, \eta_{d-1})$, we construct a modified gram matrix M_d with dimension $d \times d$, and matrix T_d with dimension $(d+1) \times d$ by the three terms recurrence of the basis t_i. M_d can be factorized to be $M_d =$

LL^T by the Cholesky factorization. The parameters H can be computed by a least squares problem of the formula:

$$min\|l_{11}e_1 - F_dH\| \tag{4}$$

With the definition of $\Omega_i \in R^{N\times s}$ by $\Omega_i = t_i(A)R_0$, we can obtain the Equation (5), and in the end iteration (6).

$$\Omega_{i+1} = \frac{1}{\beta_{i+1}}(A\Omega_i - \alpha_i\Omega_i - \delta_i\Omega_{i-1}) \tag{5}$$

$$X_n = X_0 + \mathcal{P}_d(A)R_0 = X_0 + \sum_{i=1}^{n-1}\eta_i\Omega_i \tag{6}$$

The pre-treatement of this method to obtain the parameters $A_d = (\alpha_0, \cdots, \alpha_{d-1})$, $B_d = (\beta_1, \cdots, \beta_d)$, $\Delta_d = (\delta_1, \cdots, \delta_{d-1})$, and $H_d = (\eta_0, \cdots, \eta_{d-1})$ is presented in Algorithm 3, where A is a $n \times n$ matrix, B represents the multi-vector of RHSs, d is the degree of Least Squares polynomial, Λ_r the collection of approximate eigenvalues, a, c, b the required parameters to fix an ellipse in the plan, with a the distance between the vertex and centre, c the centre position and b the focal distance. The iterative reccurence implementation of Equation (5) and (6) using the parameters gotten from the pre-treatement procedure to construct the restarted residual for BGMRES by B-LSP is given in Algorithm 4. In this algorithm, X_0 is the temporary solution in BGMRES before performing the restart. Compared with Least Squares polynomial method for single RHS, the difference in B-LSP is to replace the SpMV in each iteration step with SpGEMM, as shown in Equation (6).

Algorithm 3 Least Square Polynomial Pre-treatement

function LS(*input*: A, B, d, Λ_r, *output*: A_d, B_d, Δ_d, H)
  construct the convex hull C by Λ_r
  construct $ellispe(a, c, b)$ by the convex hull C
  compute parameters A_d, B_d, Δ_d by $ellispe(a, c, b)$
  construct matrix T $(d + 1) \times d$ matrix by A_d, B_d, Δ_d
  construct Gram matrix M_d by Chebyshev polynomials basis
  Cholesky factorization $M_d = LL^T$
  $F_d = L^TT$
  H_d satisfies min $\|l_{11}e_1 - F_dH\|$
end function

3.3. Analysis

Suppose that the computed convex hull by B-LSP contains eigenvalues $\lambda_1, \cdots, \lambda_m$, the restarted residual for BGMRES generated by B-LSP for solving Equation (1) can be divided into two parts:

$$R_n = \sum_{i=1}^{m}\sum_{j=1}^{s}\rho((\mathcal{R}_d^{(j)})(\lambda_i)^\iota)u_i + \sum_{i=m+1}^{n}\sum_{j=1}^{s}\rho((\mathcal{R}_d^{(j)})(\lambda_i)^\iota)u_i \tag{7}$$

The first part is constructed with the m known eigenvalues used to compute the convex hull in B-LSP Component, and the second part represents the residual with unknown eigenpairs.

Algorithm 4 Update BGMRES residual by LS Polynomial

function LSUpdateResidual(*input*:$A, B, A_d, B_d, \Delta_d, H_d$)
  Get X_0, which is temporary solution in BGMRES
  $R_0 = B - AX_0$, $\Omega_1 = R_0$
  for $k = 1, 2, \cdots, l$ **do**
    for $i = 1, 2, \cdots, d - 1$ **do**
      $\Omega_{i+1} = \frac{1}{\beta_{i+1}}[A\Omega_i - \alpha_i\Omega_i - \delta_i\Omega_{i-1}]$
      $X_{i+1} = X_i + \eta_{i+1}\Omega_{i+1}$
    end for
  end for
  Update GMRES restarted residual by X_d
end function

In the practical implementation, for each time preconditioning by the B-LSP method, it is often repeated for several times to improve its acceleration of convergence, that is the meaning of parameter l in Equation (7). The B-LSP preconditioning applies R_d as a deflation vector for each time restart of BGMRES. The first part in Equation (7) is small since the B-LSP finds R_d minimizing $|\mathcal{R}_d(\lambda)|$ in the convex hull, but not with the second part, where the residual will be rich in the eigenvectors associated with the eigenvalues outside H_k. As the number of approximated eigenvalues k increasing, the first part will be much closer to zero, but the second part keeps still large. This results in an enormous increase of restarted BGMRES preconditioned vector norm. Meanwhile, when BGMRES restarts with the combination of some eigenvectors, the convergence will be faster even if the residual is enormous, and the convergence of BGMRES can still be significantly accelerated.

m-UCGLE is a combination of different methods. Thus it has a large number of parameters, which have impacts on the convergence. These parameters are listed and classified according to their relations with different components as follows:

I. BGMRES Component
  * m_g: BGMRES Krylov Subspace size
  * ϵ_g: relative tolerance for BGMRES convergence test
  * P_g: number of computing units for each BGMRES
  * l: number of times that polynomial applied on the residual before taking account into the new eigenvalues
  * L: number of BGMRES restarts between two times of B-LSP preconditioning
  * s: number of RHSs

II. s-KS Component
  * m_a: s-KS Krylov subspace size
  * r: number of eigenvalues required
  * ϵ_a: tolerance for the s-KS convergence test
  * P_a: number of computing units for s-KS
  * σ: shifted value

III. B-LSP Component
  * d: Polynomial degree of B-LSP

4. Manager Engine Implementation

As presented in Section 2.3.2, the former implementation of manager engine in [1] based on MPI_Split cannot meet the

requirement of m-UCGLE. Thus, in order to extend UCGLE method to solve non-Hermitian linear systems and to reduce the global communication and synchronization points, we design and implement a new manager engine for m-UCGLE. As shown in Figure 2, the new engine allows creating a number of different computing components at the same time. Suppose that we have allocated n_g BGMRES Components, n_k s-KS Components and 1 B-LSP Component. The exact implementation for s-KS, B-LSP, BGMRES Components and manager process are respectively given in Algorithms 5, 6, 7 and 8. Denote the BGMRES Components as BGMRES[k] with $k \in 1, 2, \cdots, n_g$, and the s-KS Components as s-KS[q] with $q \in 1, 2, \cdots, n_a$. The matrix B in Equation (1) can be decomposed as:

$$B = [B_1, B_2, \cdots, B_k, \cdots, B_{n_g}] \tag{8}$$

Each BGMRES[k] will solve the linear systems with multiple RHSs B_k, which is a subgroup of B:

$$AX_k = B_k \tag{9}$$

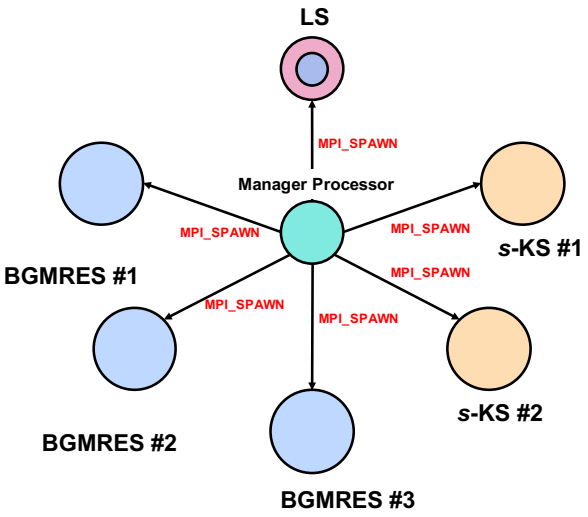

Figure 2: Manager Engine Implementation for m-UCGLE.

Table 1 gives the comparison of memory and communication complexity of SpGEMM operation inside m-UCGLE and BGMRES with the same number of RHSs. The factors n, nnz, s, P_g and n_g represent respectively the matrix size, the number of non-zero entries of matrix, the number of multiple RHSs, the total number of computing units for BGMRES Components, and the number of BGMRES Components allocated by the manager engine of m-UCGLE. The average memory requirement for BGMRES on each computing unit is $O(\frac{nnz(1+s)}{P_g})$. For m-UCGLE, the matrix is duplicated n_g times into different allocated linear solvers. Thus the required memory to store the matrix should be scaled with the factor n_g comparing with BGMRES. Due to the localization of computation inside m-UCGLE, the total number global communication can be reduced with a factor $\frac{1}{n_g}$ comparing with BGMRES. In practice, the selection of the number to allocate the BGMRES Components should

Table 1: Memory and Communication Complexity Comparison between m-UCGLE and BGMRES.

	m-UCGLE	BGMRES	ratio
Memory	$O(\frac{nnz(n_g+s)}{P_g})$	$O(\frac{nnz(1+s)}{P_g})$	$\frac{n_g+s}{1+s}$
Communication	$O(\frac{nnzsP_g}{n_g})$	$O(nnzsP_g)$	$\frac{1}{n_g}$

Algorithm 5 s-KS Component

```
1:  function s-KS-EXEC(input: A, m_a, ν, r, ε_a)
2:      while exit==False do
3:          s-KS(A, r, m_a, ν, ε_a, output: Λ_r)
4:          Send (Λ_r) to LS
5:          if Recv (exit == TRUE) then
6:              Send (exit) to LS Component
7:              stop
8:          end if
9:      end while
10: end function
```

make a balance between the increase of memory requirement and the reduction of global communication.

Here we present in detail the workflow of this new engine. In the beginning, the manager will simultaneously allocate the required number of three kinds of computing components. For each BGMRES[k], it will load a full matrix A and its related subgroup B_k, and then start to solve Equation (9) separately. Meanwhile, each s-KS[q] load a full matrix A from local, and start to find the required part of eigenvalues of A, through the s-KS method, using different parameters such as the shift value σ_q, the Krylov subspace size $(m_a)_q$, etc. If the eigenvalues of the required number are approximated on s-KS[q], these values will be asynchronously sent to the manager process. The manager process will always check if new eigenvalues are available from different s-KS Components, if yes, it will collect and update the new coming eigenvalues together and send them to B-LSP Component. B-LSP Component will use all the eigenvalues received from manager process to do the pre-treatment of the B-LSP, the parameters gotten will be sent back to the manager process. Immediately, these parameters will be distributed to BGMRES[k]. BGMRES[k] can use the B-LSP residual constructed by these parameters to speed up the convergence. If the exit signals from all BGMRES Components are received by manager process, it will send a signal to all other components to terminate their executions.

The allocation of a different number of computing components is implemented with MPI_SPAWN, and their asynchronous communication is assured by the MPI non-blocking sending and receiving operations between the manager process and each computing components.

Same as UCGLE, m-UCGLE has multiple levels of parallelism for distributed memory systems:

1. Coarse Grain/Component level: it allows the distribution of different numerical components, including the preconditioning part (B-LSP and s-KS) and the solving part

Algorithm 6 B-LSP Component

function B-LSP-EXEC(*input*: A, b, d)
  if Recv(Λ_r) **then**
    LS(*input*: A, b, d, Λ_r, *output*: A_d, B_d, Δ_d, H_d)
    Send $(A_d, B_d, \Delta_d, H_d)$ to GMRES Component
  end if
  if Recv ($exit == TRUE$) **then**
    stop
  end if
end function

Algorithm 7 BGMRES Component

function BGMRES-EXEC(*input*: $A, m_g, X_0, B, \epsilon_g, L, l$, *output*: X_m)
  $count = 0$
  BGMRES(*input*: A, m, X_0, B, *output*: X_m)
  if $\|B - AX_m\| < \epsilon_g$ **then**
    return X_m
    Send ($exit == TRUE$) to manager process
    Stop
  else
    if $count \mid L$ **then**
      if recv $(A_d, B_d, \Delta_d, H_d)$ **then**
        LSUpdateResidual(*input*:$A, B, A_d, B_d, \Delta_d, H_d$)
        $count++$
      end if
    else
      set $X_0 = X_m$
      $count++$
    end if
  end if
  if Recv ($exit == TRUE$) **then**
    stop
  end if
end function

(BGMRES) on different platforms or processors;

2. Medium Grain/Intra-component level, BGMRES and *s*-KS components are both deployed in parallel;
3. Fine Grain/Thread parallelism for shared memory: the OpenMP thread level parallelism, or the accelerator level parallelism if GPUs or other accelerators are available.

5. Hardware/Software Settings and Test Sparse Matrices

After the implementation of *m*-UCGLE, we test it on the supercomputer with selected test matrices. The purpose of this section is to give the details about the hardware/software settings and test sparse matrices.

5.1. Hardware and Software Settings

Experiments were obtained on the supercomputer *ROMEO*[2], a system located at University of Reims Champagne-Ardenne,

[2]https://romeo.univ-reims.fr

Algorithm 8 Manger of *m*-UCGLE with MPI Spawn

function MASTER($Input : n_g, n_a$)
  for $i = 1 : n_g$ **do**
    MPI_Spawn executable BGMRES-EXEC[i]
  end for
  for $j = 1 : n_k$ **do**
    MPI_Spawn executable s-KS-EXEC[j]
  end for
  MPI_Spawn executable B-LSP-EXEC
  for $j = 1 : n_k$ **do**
    if Recv $array[j]$ from s-KS-EXEC[j] **then**
      Add $array[j]$ to $Array$
    end if
  end for
  if $Array \neq NULL$ **then**
    Send $Array$ to B-LSP-EXEC
  end if
  if Recv $LSArray$ from B-LSP-EXEC **then**
    for $i = 1 : n_g$ **do**
      Send $LSArray$ to BGMRES-EXEC[i]
    end for
  end if
  for $i = 1 : n_k$ **do**
    if Recv $flag[i]$ for BGMRES-EXEC[i] **then**
      if $flag[i] == exit$ **then**
        $flag = true$
      else
        $flag = false$
      end if
    end if
  end for
  if $flag == true$ **then**
    Kill B-LSP-EXEC
    for $i = 1 : n_g$ **do**
      Kill BGMRES-EXEC[i]
    end for
    for $j = 1 : n_k$ **do**
      Kill s-KS-EXEC[j]
    end for
  end if
end function

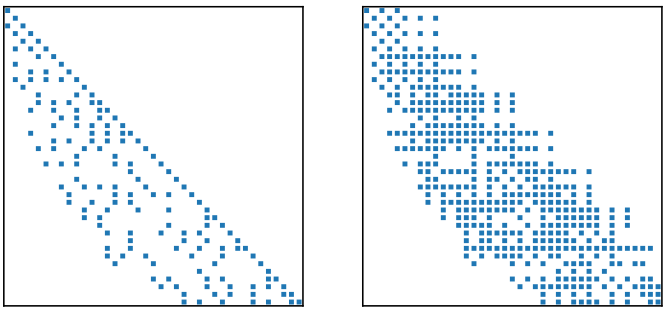

Figure 3: SMG2S example [23].

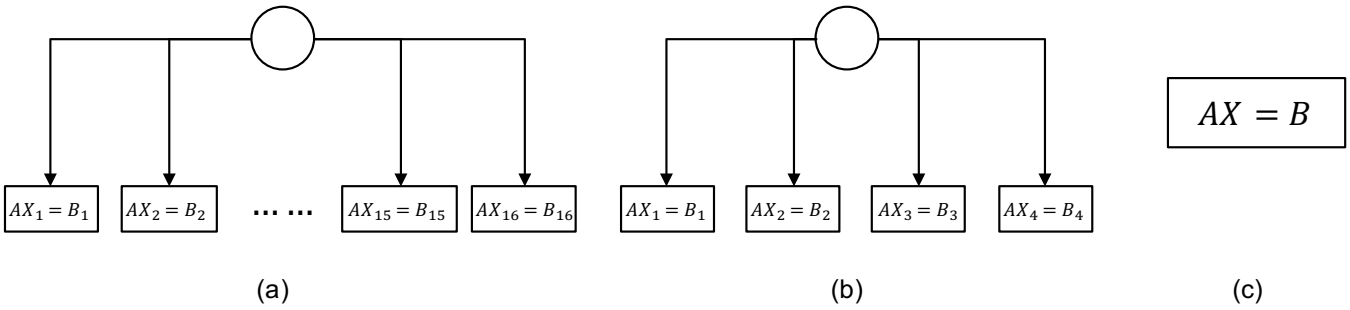

Figure 4: Different strategies to divide the linear systems with 64 RHSs into subsets: (a) divide the 64 RHSs into to 16 different components of *m*-UCGLE, each holds 4 RHSs; (b) divide the 64 RHSs into to 4 different components of *m*-UCGLE, each holds 16 RHSs; (c) One classic BGMRES to solve the linear systems with 64 RHSs simultaneously.

France. Made by Atos, this cluster relies on totally 115 the Bull Sequana X1125 hybrid servers, powered by the Xeon Gold 6132 (products formerly Skylake) and NVidia P100 cards. Each dense Bull Sequana X1125 server accommodates 2 Xeon Scalable Processors Gold bi-socket nodes, and 4 NVidia P100 cards connected with NVLink. In total, this supercomputer includes 3,220 Xeon cores and 280 Nvidia P100 accelerators.

The MPI (Message Passing Interface) used is the OpenMPI 3.1.2, all the shared libraries and binaries were compiled by gcc (version 6.3.0). The released scientific computational libraries Trilinos (version 12.12.1) and LAPACK (version 3.8.0) were also compiled and used for the implementation of *m*-UCGLE.

5.2. Test Sparse Matrices

In order to test *m*-UCGLE with matrices of high dimensions, we use the Scalable Matrix Generator with Given Spectra (SMG2S) [23, 24] to generate different test matrices. SMG2S[3] is an open source package implemented and optimized using MPI and C++ templates, which allows generating efficiently large-scale sparse non-Hermitian test matrices with customized eigenvalues or spectral distribution by users to evaluate the impacts of spectra on the convergence of linear solvers. Figure 3 gives an example of SMG2S to generate test matrix. SMG2S has good scalability and acceptable accuracy to keep the given spectra. One more important benefit of SMG2S is that the matrices are generated in parallel, the data are already allocated to different processes. These distributed data can be used directly by the users to efficiently evaluate the numerical methods without concerning the I/O operation. Various interfaces provided by SMG2S to different programming languages and computational libraries on various computer architectures make it easier to do the tests and benchmarks.

In order to generate the test matrices with given dimensions and user-defined spectral distribution, three parameters p, d and h should be specified. The definition in detail of these parameters can be referred to [23]. As in shown Fig. 3, h determine the lower band's bandwidth of generated matrices as the same value, and their upper band's bandwidth is bounded by $2pd$. In this paper, denote SMG2S($p, d, h, spec$) a collection of matrix generation suite determined by p, d, h and the prescribed spectral distribution function $spec$, which can create sparse non-Hermitian test matrices with different sizes.

[3] https://smg2s.github.io

6. Convergence Evaluation

In this section, we evaluate the numerical performance of *m*-UCGLE for solving non-Hermitian linear systems compared with conventional BGMRES.

6.1. Specific Experimental Setup

Table 2: Alternative methods for experiments, and the related number of allocated component, Rhs number per component and preconditioners.

Method	Component nb	RHS nb per component	Preconditioner
BGMRES(64)	1	64	None
m-BGMRES(16)×4	4	16	None
m-BGMRES(4)×16	16	4	None
m-UCGLE(16)×4	4	16	B-LSP
m-UCGLE(4)×16	16	4	B-LSP

In the experiments, the total number of RHSs of linear systems to be solved for each test is fixed as 64. As shown in Figure 4, we propose three strategies to divide these systems into various subgroup:

(1) 1 group with all 64 RHSs solved by classic BGMRES (shown by Figure 4(c));
(2) 4 allocated BGMRES Components in *m*-UCGLE with each holding 16 Rhs (shown by Figure 4(a));
(3) 16 allocated BGMRES Components in *m*-UCGLE with each holding 4 Rhs (shown in Figure 4(b)).

Moreover, for *m*-UCGLE with 4 or 16 allocated components, they can be applied either with or without the preconditioning of B-LSP using approximate eigenvalues. Denote the special variant of *m*-UCGLE without B-LSP preconditioning as *m*-BGMRES. *m*-BGMRES is also able to reduce the global communications through allocating multiple BGMRES components by the manager engine. Table 2 gives the naming of the five alternatives to solve linear systems with 64 RHSs and

Table 3: Iteration steps of convergence comparison (SMG2S generation suite SMG2S$(1, 3, 4, spec)$, relative tolerance for convergence test = 1.0×10^{-8}), Krylov subspace size $m_g = 40$, $s_{use} = 10$, $d = 15$, $L = 1$, *dnc* = do not converge in 5000 iteration steps).

Method	*spec*	*m*-BGMRES(4)×16	*m*-UCGLE(4)×16	*m*-BGMRES(16)×4	*m*-UCGLE(16)×4	BGMRES(64)
TEST-1	(rand(21.0, 66.0), rand(-21.0,24.0))	239	160	102	51	51
TEST-2	(rand(0.5, 3.0), rand(-0.5,2.0))	*dnc*	176	187	62	78
TEST-3	(rand(0.2, 5.2), rand(-2.5,2.5))	*dnc*	310	*dnc*	81	657
TEST-4	(rand(-5.2, -0.2), rand(-2.5,2.5))	*dnc*	320	629	99	942
TEST-5	(rand(-0.23, -0.03), rand(-0.2,0.2))	600	235	170	99	270
TEST-6	(rand(-9.3, -3.2), rand(-2.1,2.1))	80	160	85	51	38

the numbers of their allocated components and the numbers of RHSs per component.

These 64 RHSs for the tests are all generated in random with different given seed states. All the test matrices from TEST-1 to TEST-6 in Table 3 are generated by SMG2S$(1, 3, 4, spec)$, the definition of *spec* functions for different tests are shown in Table 3. For example, the *spec* of TEST-1 is given as (rand(21.0, 66.0), rand(-21.0,24.0)), the first part rand(21.0, 66.0) defines that the real parts of given eigenvalues for TEST-1 are the floating numbers generated randomly in the fixed interval [21.0, 66.0], similarly its imaginary parts are randomly generated in the fixed interval [-21.0, 24.0]. The relative tolerance for the convergence test is fixed as 1.0×10^{-8}, the Krylov subspace size m_g is given as 40 for all tests, the number of times that B-LSP applied in *m*-UCGLE l, the degree of Least Squares polynomial in B-LSP, the number of BGMRES restarts between two times of B-LSP preconditioning are respectively set as 10, 15 and 1. For *m*-BGMRES(16)×4, *m*-BGMRES(4)×16, *m*-UCGLE(16)×4 and *m*-UCGLE(16)×4, their iteration steps in Table 3 are defined as the maximal ones among their allocated components to solve the systems.

6.2. Result Analysis

The iteration steps of different methods for convergence are shown in Table 3 (*dnc* in this table significates *do not converge in 5000 iteration steps*). In this table, the blue and red cells represent respectively the worst and the best cases for each test. From Table 3, firstly we can conclude that the enlargement of the Krylov subspace by more RHSs in BGMRES is effective to accelerate the convergence for TEST-1, TEST-2, TEST-3, and TEST-6. However, this acceleration cannot always be guaranteed. Referring to TEST-4 and TEST-5, *m*-BGMRES(16))×4 converge faster than BGMRES(64). Secondly, *m*-UCGLE components converge much faster than their related *m*-BGMRES with the same number of RHSs, except in TEST-6. In the TEST-2, TEST-3, TEST-4, and TEST-5 of this table, *m*-UCGLE with 16 RHSs works even much better than BGMRES with 64 RHSs. An extreme special case in TEST-4 shows that convergence of *m*-UCGLE(4)×16 and *m*-UCGLE(16)×4 have the speedup respectively 3× and 9.5× over *m*-BGMRES(16).

In conclusion, for most tests, the method that converges the fastest is *m*-UCGLE(16)×4. The combination of enlarging the search space by enough number of RHSs and preconditioning by B-LSP makes substantial acceleration on the convergence of solving linear systems. Moreover, compared with classic BGMRES, due to the preconditioning of B-LSP, *m*-UCGLE with less RHSs and smaller search space can still have a better acceleration of the convergence. The potential damages caused by the localization of computation and reduction of global communications with less RHSs of each component in *m*-UCGLE can be covered by the B-LSP preconditioning.

7. Parallel Performance Evaluation

The iteration step for convergence is not the only concern about the iterative methods. After the convergence comparison of *m*-UCGLE and BGMRES, in this section, we evaluate its strong scalability and performance on supercomputer *ROMEO*.

7.1. Strong Scalability Evaluation

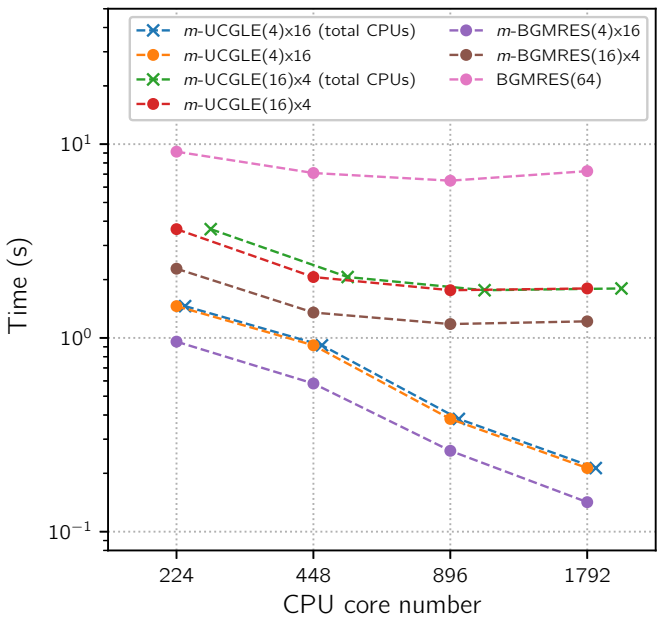

Figure 5: Strong scalability test on CPUs of solving time per iteration for *m*-BGMRES(4)×16, *m*-UCGLE(4)×16, *m*-BGMRES(16)×4, *m*-UCGLE(16)×4, BGMRES(64); test matrix size is 1.792×10^6; X-axis refers to the total number of CPUs from 224 to 1792; Y-axis refers to the average execution time per iteration. A base 2 logarithmic scale is used for all X-axis, and a base 10 logarithmic scale is used for all Y-axis.

Table 4: Consumption time (s) comparison on CPUs (SMG2S generation suite SMG2S$(1, 3, 4, spec)$, the size of matrices = 1.792×10^6, relative tolerance for convergence test = 1.0×10^{-8}), Krylov subspace size $m_g = 40$, $l = 10$, $d = 15$, $L = 1$, *dnc* = do not converge in 5000 iteration steps).

Method	*spec*	*m*-BGMRES(4)×16	*m*-UCGLE(4)×16	*m*-BGMRES(16)×4	*m*-UCGLE(16)×4	BGMRES(64)
TEST-1	(rand(21.0, 66.0), rand(-21.0,24.0))	34.2	35.3	133.9	98.9	362.8
TEST-2	(rand(0.5, 3.0), rand(-0.5,2.0))	*dnc*	40.9	231.3	111.5	580.6
TEST-3	(rand(0.2, 5.2), rand(-2.5,2.5))	*dnc*	66.0	*dnc*	145.8	522.5
TEST-4	(rand(-5.2, -0.2), rand(-2.5,2.5))	*dnc*	68.2	768.3	178.2	6829.3
TEST-5	(rand(-0.23, -0.03), rand(-0.2,0.2))	132.3	50.1	209.4	120.5	1959.5
TEST-6	(rand(-9.3, -3.2), rand(-2.1,2.1))	11.4	34.1	87.8	91.7	275.8

One important concern of BGMRES is the time cost per iteration, due to the communication bound of SpGEMM. In order to evaluate the parallel performance of *m*-UCGLE on large clusters, we compare the strong scaling of *m*-BGMRES(4)×16, *m*-UCGLE(4)×16, *m*-BGMRES(16)×4, *m*-UCGLE(16)×4 and BGMRES(64) by their average time cost per iteration.

For the strong scaling evaluation on CPUs, the methods tested are set up the same as in Section 6. The test matrix is generated by SMG2S$(1, 3, 4, spec)$ with size fixed as 1.792×10^6. No larger matrices are tested, due to the memory limitation during the Arnoldi projection of BGMRES. The Krylov subspace size m_g for all methods is set as 40. The average time cost for these methods is computed by 100 iterations. Time per iteration is suitable for demonstrating scaling behavior. The total CPU core number for B-GMRES(64) and all BGMRES Components in *m*-UCGLE and *m*-BGMRES ranges from 224 to 1792. Thus for each BGMRES Component of *m*-BGMRES(4)×16 and *m*-UCGLE(4)×16, the number ranges from 14 to 112. Similarly, for each BGMRES Component of *m*-BGMRES(16)×4 and *m*-UCGLE(16)×4, this number ranges from 56 to 448. All the tests allocate only 1 *s*-KS Component which always has the same number of CPU cores with each BGMRES Component inside *m*-UCGLE.

In Figure 5, we can conclude that the strong scaling of *m*-BGMRES(4)×16 and *m*-UCGLE(4)×16 perform very well, the strong scalability of the rest are bad, especially BGMRES(64). In the beginning, the scalability of *m*-BGMRES(16)×4 and *m*-UCGLE(16)×4 is good, but it turns bad quickly with the increase of CPU number. It is demonstrated that the properties of m-UCGLE to promote the asynchronous communication and localize of computation can improve significantly the parallel performance of BGMRES for solving systems with multiple RHSs. Additionally, for the *m*-BGMRES and *m*-UCGLE with the same number of RHSs, the time per iteration of former is a little less the latter, since *m*-UCGLE introduces the iterative operations (SpGEMM) by B-LSP preconditioning.

Since *m*-UCGLE uses additional computing units for other components especially *s*-KS Component, it is unfair only to compare the scaling performance that total CPU number of all BGMRES Components in *m*-UCGLE equals to these numbers of *m*-GMRES and BGMRES(64). Thus we plot two more curves of *m*-UCGLE(4)×16 and *m*-UCGLE(16)×14 with all their CPU numbers (including the CPU of *s*-KS Component) in Figure 5. The two additional curves are respectively the blue and green ones with the marker set as the cross. It is shown that *m*-UCGLE(4)×16 and *m*-UCGLE(16)×4 can still have respectively up to 35× and 4× speedup per iteration against BGMRES(64).

7.2. Time Consumption Evaluation

After the evaluation of parallel performance, we compare the time consumption of all methods to solve large-scale linear systems with multiple RHSs on CPUs. The test matrices are generated by SMG2S$(1, 3, 4, spec)$, the same as the convergence evaluation in Section 6. The size of matrices are all 1.792×10^6 for the experiments on CPUs, the total numbers of CPU cores for different methods (either BGMRES Components in *m*-UCGLE and *m*-GMRES or conventional BGMRES) are respectively fixed as 1792, and the Krylov subspace size m_g is set as 40. The results on CPUs is given in Table 4, where the blue and red cells represent respectively the worst and the best cases for each test.

We can find that for the TEST-2, TEST-3, TEST-4, TEST-4, *m*-UCGLE(4)×16 take the least time to converge. For TEST-1 and TEST-6, *m*-BGMRES(4)×16 takes a little less time than *m*-UCGLE(4)×16 to get the convergence. For an extremely special case TEST-4, *m*-UCGLE(4)×16 has about 100× speedup in time consumption over BGMRES(64) to solve the linear systems with 64 RHSs.

7.3. Performance Analysis

In conclusion, *m*-UCGLE(4)×16 and *m*-GMRES(4)×16 with the most decrease of global communications have the best strong scaling performance. *m*-UCGLE cost a little more time per iteration compared with *m*-BGMRES with the same number of RHSs, but this might be made up by its decrease of iterative steps with B-LSP preconditioning. The experiments in this section demonstrate the benefits of *m*-UCGLE to reduce global communication and promote the asynchronization. Two more important points that cannot be concluded from the experiments in this paper are:

(1) The increase of memory requirement should be considered when dividing the whole RHSs into subsets;
(2) The number of RHSs per component of *m*-UCGLE to enlarge the search space and the computation time per iteration should be balanced to achieve the best performance.

7.4. Requirement of Workflow Environment for UC Approach

After the validation of the numerical and parallel performance of *m*-UCGLE, a major difficulty to profit from UC methods including *m*-UCGLE is to implement the manager engine which can well handle their fault tolerance, load balance, asynchronous communication of signals, arrays and vectors, the management of different computing units such as GPUs, etc. In this paper, we tried to give a naive implementation of the engine to support the management tasks on the homogeneous platforms based on MPI_SPAWN and MPI non-blocking sending and receiving functionalities. Further development of this engine should be done to support the manager of the computing algorithms and their asynchronous communication on more different computer architectures and platforms, such as the multi-GPU connected with NVLink on *ROMEO*. The stability of this implementation of the engine cannot always be guaranteed. Thus we are also thinking about to select the suitable workflow/task based environments to manage all these aspects in the UC approach. YML[4] is a good candidate, which is a workflow environment to provide the definition of the parallel application independently from the underlying middleware used. The special middleware and workflow scheduler provided by YML allows defining the dependencies of tasks and data on the supercomputers [25]. YML, including its interfaces and compiler to various programming languages and libraries, will facilitate the implementation of UC based methods with different numerical components. We plan to replace the manager engine of *m*-UCGLE by YML in the future.

8. Conclusion and Perspectives

This paper presents *m*-UCGLE, an extension of distributed and parallel method UCGLE to solve large-scale non-Hermitian sparse linear systems with multiple RHSs on modern supercomputers. *m*-UCGLE is implemented with three kinds of computational components which communicate by the asynchronous communication. A special engine is proposed to manger the communication and allocate multiple different components at the same time. *m*-UCGLE is able to accelerate the convergence, minimize the global communication, cover the synchronous points for solving linear systems with multiple RHSs on large-scale platforms. The experiments on supercomputers prove the good numerical and parallel performance of this method.

Various parameters have impacts on the convergence. Thus, the auto-tuning scheme is required in the next step, where the systems can select the dimensions of Krylov subspace, the numbers of eigenvalues to be computed, the degrees of Least Squares polynomial according to different linear systems and cluster architectures. The fault tolerance and reusability of *m*-UCGLE and the influence of multiple *s*-KS components with different shifted values need to be evaluated. Parallel performance on heterogeneous clusters (e.g., multi-GPU) should be tested with further development of the manager engine, and workflow based model can be applied to replace the proposed manager engine in the future.

[4] http://yml.prism.uvsq.fr/

Acknowledgment

The authors would like to thank ROMEO HPC Center Champagne Ardenne for their support in providing the use of cluster *ROMEO*. This work is funded by the MYX project of French National Research Agency (ANR) (Grant No. ANR-15-SPPE-003) under the SPPEXA framework.

References

[1] X. Wu, S. G. Petiton, A distributed and parallel asynchronous unite and conquer method to solve large scale non-hermitian linear systems, in: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, ACM, 2018, pp. 36–46.

[2] T. Sakurai, H. Tadano, Y. Kuramashi, Application of block krylov subspace algorithms to the wilson–dirac equation with multiple right-hand sides in lattice qcd, Computer Physics Communications 181 (1) (2010) 113–117.

[3] Y. Nakamura, K.-I. Ishikawa, Y. Kuramashi, T. Sakurai, H. Tadano, Modified block bicgstab for lattice qcd, Computer Physics Communications 183 (1) (2012) 34–37.

[4] P. Fiebach, A. Frommer, R. Freund, Variants of the block-qmr method and applications in quantum chromodynamics, in: 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Vol. 3, 1997, pp. 491–496.

[5] M. Malhotra, R. W. Freund, P. M. Pinsky, Iterative solution of multiple radiation and scattering problems in structural acoustics using a block quasi-minimal residual algorithm, Computer methods in applied mechanics and engineering 146 (1-2) (1997) 173–196.

[6] G. Barbella, F. Perotti, V. Simoncini, Block krylov subspace methods for the computation of structural response to turbulent wind, Computer Methods in Applied Mechanics and Engineering 200 (23-24) (2011) 2067–2082.

[7] F. G. Ferraz, J. M. C. Dos Santos, Block-krylov component synthesis and minimum rank perturbation theory for damage detection in complex structures, Proceeding of the IX DINAME, Florianópolis-SC-Brazil (2001) 329–334.

[8] B. Nour-Omid, R. W. Clough, Short communication block lanczos method for dynamic analysis of structures, Earthquake engineering & structural dynamics 13 (2) (1985) 271–275.

[9] E. Agullo, L. Giraud, Y.-F. Jing, Block gmres method with inexact breakdowns and deflated restarting, SIAM Journal on Matrix Analysis and Applications 35 (4) (2014) 1625–1651.

[10] J. Demmel, M. Hoemmen, M. Mohiyuddin, K. Yelick, Avoiding communication in sparse matrix computations, in: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, IEEE, 2008, pp. 1–12.

[11] M. Hoemmen, Communication-avoiding krylov subspace methods, Ph.D. thesis, UC Berkeley (2010).

[12] E. C. Carson, Communication-avoiding krylov subspace methods in theory and practice, Ph.D. thesis, UC Berkeley (2015).

[13] P. Ghysels, T. J. Ashby, K. Meerbergen, W. Vanroose, Hiding global communication latency in the gmres algorithm on massively parallel machines, SIAM Journal on Scientific Computing 35 (1) (2013) C48–C71.

[14] H. Morgan, M. G. Knepley, P. Sanan, L. R. Scott, A stochastic performance model for pipelined krylov methods, Concurrency and Computation: Practice and Experience 28 (18) (2016) 4532–4542.

[15] S. Cools, W. Vanroose, The communication-hiding pipelined bicgstab method for the parallel solution of large unsymmetric linear systems, Parallel Computing 65 (2017) 1–20.

[16] B. Vital, Etude de quelques méthodes de résolution de problemes linéaires de grande taille sur multiprocesseur, Ph.D. thesis, Rennes 1 (1990).

[17] Y. Saad, Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems, SIAM Journal on Numerical Analysis 24 (1) (1987) 155–169.

[18] Y. Saad, M. H. Schultz, Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on scientific and statistical computing 7 (3) (1986) 856–869.

[19] Y. Saad, Numerical methods for large eigenvalue problems: revised edition, Vol. 66, Siam, 2011.

[20] D. C. Sorensen, Implicitly restarted arnoldi/lanczos methods for large scale eigenvalue calculations, in: Parallel Numerical Algorithms, Springer, 1997, pp. 119–165.

[21] G. W. Stewart, A krylov–schur algorithm for large eigenproblems, SIAM Journal on Matrix Analysis and Applications 23 (3) (2002) 601–614.

[22] N. Emad, S. Petiton, Unite and conquer approach for high scale numerical computing, Journal of Computational Science 14 (2016) 5–14.

[23] X. Wu, S. Petiton, Y. Lu, A parallel generator of non-hermitian matrices computed from given spectra, in: VECPAR 2018: 13th International Meeting on High Performance Computing for Computational Science, 2018.

[24] X. WU, SMG2S Manual v1.0, Technical report, Maison de la Simulation (Sep. 2018).

[25] O. Delannoy, Yml: un workflow scientifique pour le calcul haute performance, Ph.D. thesis, Université de Versailles Saint-Quentin, France (2008).